Requested Patent: JP7210424A

Title: SOFTWARE TEST SUPPORTING SYSTEM ;

Abstracted Patent: JP7210424 ;

Publication Date: 1995-08-11 ;

Inventor(s): HIRAYAMA MASAYUKI; others: 02 ;

Applicant(s): TOSHIBA CORP ;

Application Number: JP19940002810 19940114 ;

Priority Number(s): ;

IPC Classification: G06F11/28 ; G06F9/06 ;

Equivalents:

ABSTRACT:

PURPOSE:To provide a software test supporting system capable of easily executing an efficient software test.

CONSTITUTION:A source code 1 prepared by a structured programming procedure is sorted into plural unit routines by a source code analyzing means 10 and calling relation among respective unit routines is also analyzed. A test case preparing means 20 prepares plural test cases covering all calling patterns based upon the calling relation of respective analyzed unit routines. since the test cases can be automatically prepared based on the source code 1, the test cases can be more simply prepared as compared with a convential system for preparing test cases based on the specifications of software described by natural language such as Japanese, the status transition specifications of the software, etc.

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号

特開平7－210424

(43)公開日　平成７年(1995)８月11日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　11/28	３４０　Ａ	9290－5B		
9/06	５４０	9367－5B		

審査請求　未請求　請求項の数８　ＯＬ　（全 15 頁）

(21)出願番号　特願平6－2810

(22)出願日　平成６年(1994)１月14日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　平山　雅之
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

(72)発明者　岡安　二郎
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

(72)発明者　深谷　哲司
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

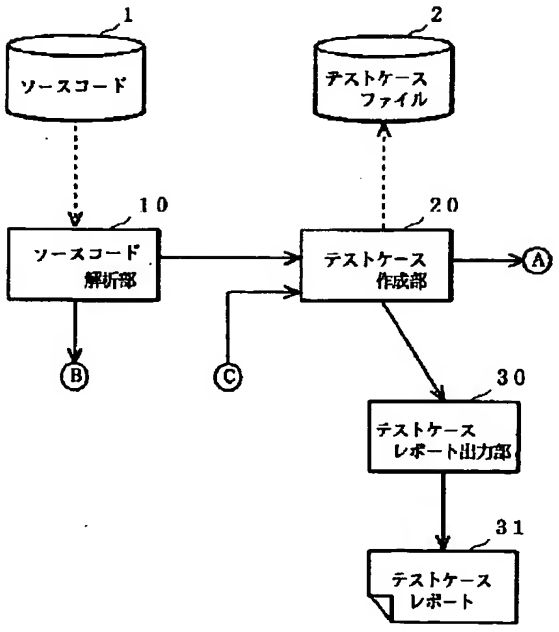
(74)代理人　弁理士　木内　光春

(54)【発明の名称】　ソフトウェアテスト支援システム

(57)【要約】

【目的】　効率的なソフトウェアテストを容易に行うことのできるソフトウェアテスト支援システムを提供することを目的とする。

【構成】　構造化プログラミング手法を用いて作成されたソースコードは、ソースコード解析手段によって複数の単位ルーチンに分類され、さらに、各単位ルーチンの呼び出し関係の解析が行われる。テストケース作成手段では、解析された各単位ルーチンの呼び出し関係に基づいて全ての呼び出しパターンを網羅する複数のテストケースが作成される。このように、ソースコードに基づいてテストケースを自動的に作成することが可能となるため、日本語などの自然言語で記述されたソフトウェアの仕様書やソフトウェアの状態遷移仕様などをベースにテストケースを作成する従来の方式に比べて、より簡便にテストケースを作成することが可能となる。



1

【特許請求の範囲】

【請求項1】　構造化プログラミング手法によって作成されたソースコードを解析してソフトウェアのテストの支援を行うソフトウェアテスト支援システムにおいて、

ソースコードを複数の単位ルーチンに分類し、各単位ルーチンの呼び出し関係を解析するソースコード解析手段と、

前記ソースコード解析手段で解析された各単位ルーチンの呼び出し関係に基づいて、全ての呼び出しパターンを網羅する複数のテストケースを作成するテストケース作成手段とを備えることを特徴とするソフトウェアテスト支援システム。

【請求項2】　前記テストケース作成手段で作成された複数のテストケースの一覧を帳票形式で出力するテストケースレポート出力手段とを備えることを特徴とする請求項1記載のソフトウェアテスト支援システム。

【請求項3】　前記ソースコードの各単位ルーチンにテストプローブを挿入するテストプローブ挿入手段と、

前記テストプローブ挿入手段によってテストプローブが挿入された前記ソースコードをコンパイル・リンクしてロードモジュールを作成するロードモジュール作成手段と、

前記テストケース作成手段で作成された各テストケースに基づいて、前記ロードモジュール作成手段で作成したロードモジュールを実行するテスト実行手段とを備え、

前記テストプローブは、少なくとも所定の履歴ファイルをオープンする命令と、この履歴ファイルに挿入対象の単位ルーチンを特定する番号を書き込む命令とを含んだ命令群であることを特徴とする請求項1または請求項2に記載のソフトウェアテスト支援システム。

【請求項4】　前記テスト実行手段でのロードモジュールの実行によって書き込まれた前記履歴ファイルを分析して、前記ソースコードの全単位ルーチンの中の何パーセントの単位ルーチンがこの履歴ファイルに書き込まれているか検出すると共に、未実行のモジュールを検出するモジュール網羅率評価手段を備えることを特徴とする請求項3記載のソフトウェアテスト支援システム。

【請求項5】　前記モジュール網羅率評価手段で検出されたモジュール網羅率および未実行モジュールの一覧を帳票形式で出力するモジュール網羅率出力手段とを備えることを特徴とする請求項4記載のソフトウェアテスト支援システム。

【請求項6】　前記テスト実行手段でのロードモジュールの実行によって書き込まれた前記履歴ファイルを分析して、前記テストケース作成手段で作成された全テストケースの中の何パーセントのテストケースがこの履歴ファイルに書き込まれているか検出すると共に、未実行のテストケースを検出するテストケース網羅率評価手段を備えることを特徴とする請求項3記載のソフトウェアテスト支援システム。

2

【請求項7】　前記テストケース網羅率評価手段で検出されたテストケース網羅率および未実行テストケースの一覧を帳票形式で出力するテストケース網羅率出力手段とを備えることを特徴とする請求項6記載のソフトウェアテスト支援システム。

【請求項8】　バグ修正またはバージョンアップによる修正が行われたソースコードを複数の単位ルーチンに分類する修正ソースコード解析手段と、

前記ソースコード解析手段で分類された単位ルーチンと前記修正ソースコード解析で分類された単位ルーチンとを比較して、削除された単位ルーチンおよび追加された単位ルーチンを検出する第1の修正単位ルーチン検出手段と、

修正前後の前記ソースコードを比較して、前記修正ソースコード解析で分類された単位ルーチンの中で修正された単位ルーチンを検出する第2の修正単位ルーチン検出手段と、

前記テストケース作成手段で作成されたテストケースの内、前記第1の修正単位ルーチン検出手段で検出された追加単位ルーチンを対象とするテストケースと前記第2の修正単位ルーチン検出手段で検出された修正単位ルーチンを対象とするテストケースとを特定する再実行対象テストケース特定手段とを備えることを特徴とする請求項1から請求項7までのいずれかに記載のソフトウェアテスト支援システム。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、ソフトウェア開発において、その動作テストを実施する際に利用されるソフトウェアテスト支援システムに関し、特に、構造化設計手法を利用してC言語等の高級言語を用いて開発されたアプリケーション・ソフトウェアの動作テストに利用されるソフトウェアテスト支援システムに関する。

【0002】

【従来の技術】従来、ソフトウェアの動作テストは、「テストケース作成の検討が不十分である。」、「テストケースを確実に試験していない。」といった2つの要因によりテストが不十分になることが多い。

【0003】通常のソフトウェア動作テストは、日本語などの自然言語で記述されたソフトウェアの仕様書をもとに、手作業でテストケースを作成し、このテストケースに基づいて行っている。しかしながら、このテストケース作成の作業は元となるソフトウェアの仕様書の記述内容（レベル）やテストケース作成を検討する個人の能力に依存する場合が多く、テストケースの質の低下を招いていた。

【0004】一方、バグ修正あるいはバージョンアップに伴うソフトウェアの修正では、修正前のソフトウェア作成の際に行われた動作テストを元にしてテストが行われることが多い。しかしながら、この手法によって行わ

3

れるテストでも、対象とするソフトウェアに関するテストケースとして十分であるかは定かではない。修正前のソフトウェアに対して新規に機能追加等を実施した場合には、その機能追加部分に対する新たなテストが必要となるからである。

【０００５】このような問題を解決する手法として、状態遷移仕様から自動的にテストケースを作成する方式が提案されている。このテストケースの自動生成方式については、以下の文献に詳細に記載されている。

【０００６】「状態遷移仕様からのテストデータ生成方式」
情報処理学会第４２回　全国大会論文集　ｐｐ５－２１８　岡安　他

この文献に記載されたテストケースの自動生成方式を概説すると、状態遷移仕様に基づいたソフトウェアの動作パターン（動作列）をテストケースとして利用するものである。具体的には、状態遷移モデルにおいて記述される状態からの遷移を引き起こすイベントに着目して、発生するイベントを順次並べたイベント系列を作成し、これによって起こるシステムの状態遷移を動作パターンとしてテストケースに利用するものである。

【０００７】しかしながら、この生成方式は状態遷移仕様がソフトウェア作成の際に作成されていることが前提となるため、その適用範囲は極めて狭く実用的ではない。

【０００８】また、テストの十分性については、ソースコードが一連の動作テストでどの程度網羅的に実行されているかを示すテストカバレジを計測することによって十分性を評価する試みが従来より行われている（参考文献：玉井哲雄　「ソフトウェアのテスト技法」　ｐ３４，３５　共立出版）。しかしながら、テストカバレジは、個々のモジュール内に含まれる命令や条件分岐等の通過率を計測するものが殆どであり、ソフトウェアの詳細部まで入り込むため、システムテストやソフトウェア・システムテスト等の上位レベルのテストの十分性評価には適していなかった。

【０００９】さらに、ソフトウェアの動作テストでは、対象となるソースコードが修正・変更された場合には、再度、テストケースについての検討が必要となる。特に構造化設計技法を利用して作成されたソフトウェアでは、ソースコードを構成する一部のモジュールを修正・変更した場合の影響が、別の部分のモジュールに影響を及ぼすケースが多く、単純に修正・変更したモジュール部分のみを再テストすればよいとは限らないからである。

【００１０】

【発明が解決しようとする課題】上述したように、従来においては、ソフトウェアテストを実施する上でのテストケースの作成方式が確立されておらず、また、テストの十分性評価手法も片寄ったものであり問題であった。

4

このため効率的にテストケースを生成し、また、このテストケースに従ってテストを実施した場合のテストの十分性評価を容易に行うことを可能とするソフトウェアテスト支援システムが望まれていた。

【００１１】本発明は、かかる従来の問題を解決するものであり、試験対象ソフトウェアのソースコードから自動的にテストケースを作成し、また、テスト実施結果とこのテストケースを比較評価し、モジュール・カバレジ計測及び未試験モジュール特定などによりテストの十分性を判定するシステムを提供することを目的とする。

【００１２】さらに、本発明は、対象となるソフトウェアが修正・変更を受けた場合にも、修正前のソースコードおよびテストケースと対比して、容易に修正ソフトウェアに対するテストケース生成等の支援を行うシステムを提供することを目的とする。

【００１３】

【課題を解決するための手段】上記課題を解決するために、本発明のソフトウェアテスト支援システムは、

（ａ）ソースコードを複数の単位ルーチンに分類し、各単位ルーチンの呼び出し関係を解析するソースコード解析手段と、（ｂ）ソースコード解析手段で解析された各単位ルーチンの呼び出し関係に基づいて、全ての呼び出しパターンを網羅する複数のテストケースを作成するテストケース作成手段とを備える。

【００１４】また、（ｃ）テストケース作成手段で作成された複数のテストケースの一覧を帳票形式で出力するテストケースレポート出力手段を備えていてもよい。

【００１５】さらに、（ｄ）ソースコードの各単位ルーチンにテストプローブを挿入するテストプローブ挿入手段と、（ｅ）テストプローブ挿入手段によってテストプローブが挿入されたソースコードをコンパイル・リンクしてロードモジュールを作成するロードモジュール作成手段と、（ｆ）テストケース作成手段で作成された各テストケースに基づいて、ロードモジュール作成手段で作成したロードモジュールを実行するテスト実行手段とを備えていてもよい。この場合、テストプローブは、少なくとも所定の履歴ファイルをオープンする命令と、この履歴ファイルに挿入対象の単位ルーチンを特定する番号を書き込む命令とを含んだ命令群である。

【００１６】さらにまた、（ｇ）テスト実行手段でのロードモジュールの実行によって書き込まれた履歴ファイルを分析して、ソースコードの全単位ルーチンの中の何パーセントの単位ルーチンがこの履歴ファイルに書き込まれているか検出すると共に、未実行のモジュールを検出するモジュール網羅率評価手段を備えていてもよい。この場合には、（ｈ）モジュール網羅率評価手段で検出されたモジュール網羅率および未実行モジュールの一覧を帳票形式で出力するモジュール網羅率出力手段とを備えていてもよい。

【００１７】さらにまた、（ｊ）テスト実行手段でのロ

5

ードモジュールの実行によって書き込まれた履歴ファイルを分析して、テストケース作成手段で作成された全テストケースの中の何パーセントのテストケースがこの履歴ファイルに書き込まれているか検出すると共に、未実行のテストケースを検出するテストケース網羅率評価手段を備えていてもよい。この場合には、（k）テストケース網羅率評価手段で検出されたテストケース網羅率および未実行テストケースの一覧を帳票形式で出力するテストケース網羅率出力手段とを備えていてもよい。

【0018】さらにまた、（l）バグ修正またはバージョンアップによる修正が行われたソースコードを複数の単位ルーチンに分類する修正ソースコード解析手段と、（m）ソースコード解析手段で分類された単位ルーチンと修正ソースコード解析で分類された単位ルーチンとを比較して、削除された単位ルーチンおよび追加された単位ルーチンを検出する第1の修正単位ルーチン検出手段と、（n）修正前後のソースコードを比較して、修正ソースコード解析で分類された単位ルーチンの中で修正された単位ルーチンを検出する第2の修正単位ルーチン検出手段と、（o）テストケース作成手段で作成されたテストケースの内、第1の修正単位ルーチン検出手段で検出された追加単位ルーチンを対象とするテストケースと第2の修正単位ルーチン検出手段で検出された修正単位ルーチンを対象とするテストケースとを特定する再実行対象テストケース特定手段とを備えていてもよい。

【0019】

【作用】本発明のソフトウェアテスト支援システムによれば、構造化プログラミング手法を用いて作成されたソースコードは、ソースコード解析手段によって複数の単位ルーチンに分類され、さらに、各単位ルーチンの呼び出し関係の解析が行われる。単位ルーチンには、タスク単位に分けたルーチンや、モジュール単位に分けたルーチンがある。ここで、タスクとは構造化プログラムにおけるソフトウェアとして一つのまとまった処理機能を実現する単位をいい、モジュールとは構造化プログラムにおけるソフトウェア構造の基本単位をいう。テストケース作成手段では、解析された各単位ルーチンの呼び出し関係に基づいて全ての呼び出しパターンを網羅する複数のテストケースが作成される。このように、試験対象ソフトウェアのソースコードから自動的にテストケースを作成することが可能となるため、日本語などの自然言語で記述されたソフトウェアの仕様書やソフトウェアの状態遷移仕様などをベースにテストケースを作成する従来の方式に比べて、より簡便にテストケースを作成することが可能となる。

【0020】また、テストケース作成手段で作成された複数のテストケースの一覧は、テストケースレポート出力手段によって帳票形式で出力される。このように出力された書類はソフトウェアテストを実施する際の参考とすることができる。

6

【0021】テストケース作成手段で作成されたテストケースに基づいてテストを行うためには、以下のように処理される。まず、テストプローブ挿入手段によってソースコードの各単位ルーチンにテストプローブが挿入される。テストプローブ挿入後のソースコードはロードモジュール作成手段によってコンパイル・リンクされてロードモジュールが作成される。このロードモジュールは、テスト実行手段で実行され、呼び出された全てのモジュールの番号が履歴ファイルに記録される。

【0022】履歴ファイルはモジュール網羅率評価手段で分析され、モジュール網羅率および未実行モジュールが検出される。このような検出を行うことにより、テストの抜けを明示的に把握することができるようになり、テストの十分性が従来に比べ飛躍的に向上する。特に、検出結果は、モジュール網羅率出力手段によって帳票形式で出力できるので、追加テスト等の作成を効率的に行うことができる。

【0023】また、履歴ファイルはテストケース網羅率評価手段でも分析され、テストケース網羅率および未実行テストケースが検出される。このような検出を行うことにより、テストの抜けを明示的に把握することができるようになり、テストの十分性が従来に比べ飛躍的に向上する。特に、検出結果は、テストケース網羅率出力手段によって帳票形式で出力できるので、追加テスト等の作成を効率的に行うことができる。

【0024】テスト時に発見されたバグ等の修正や、バージョンアップによる修正を行った場合には、修正ソースコード解析手段によって修正後のソースコードを複数の単位ルーチンに分類する。そして、第1の修正単位ルーチン検出手段で修正前後の単位ルーチンが比較され、削除された単位ルーチンおよび追加された単位ルーチンが検出される。さらに、第2の修正単位ルーチン検出手段で修正前後のソースコードが比較され、修正された単位ルーチンが検出される。このように検出された追加単位ルーチンおよび修正単位ルーチンは再実行対象テストケース特定手段に与えられ、これらの単位ルーチンを対象としたテストケースが特定される。この特定により再テストで実行すべきテストケースが判定評価され、また特に重点的にテストすべき箇所が特定できるので、効率的な再テストを実施することができる。

【0025】

【実施例】以下、本発明の一実施例を添付図面を参照して説明する。

【0026】図1～図3は、本実施例に係るソフトウェアテスト支援システムの構成を示すブロック図である。これらの図より、本実施例のソフトウェアテスト支援システムは、ソースコード1を複数のモジュール（またはタスク）に分類して各モジュールの呼び出し関係を解析するソースコード解析部10と、各モジュールの呼び出し関係に基づいて複数のテストケースを作成するテスト

ケース作成部２０と、テストケースレポート３１を出力するテストケースレポート出力部３０とを備えている。また、ソースコード１にテストプローブを挿入するテストプローブ挿入部４０と、ソースコード１をコンパイル・リンクしてロードモジュール３を作成するロードモジュール作成部５０と、ロードモジュール３を実行してテストカバレジを計測するテストカバレジ計測部６０とを備えている。

【００２７】さらに、モジュール網羅率および未実行モジュールの検出を行うモジュール網羅率評価部７０と、テストケース網羅率および未実行テストケースの検出を行うテストケース網羅率評価部８０と、テストカバレジレポート９１を出力するモジュール網羅率出力部９０と、未試験モジュール遷移パスレポート１０１を出力するテストケース網羅率出力部１００とを備えている。さらにまた、デバッグなどで修正された修正ソースコード６を複数のモジュールに分類する修正ソースコード解析部１１０と、ソースコード１の修正によるテストケースへの影響を評価するソースコード変更影響評価部１２０と、追加・変更されたモジュールを対象とするテストケースを特定する再実行対象テストケース特定部１３０とを備えている。また、ソースコード変更影響評価部１２０には、削除されたモジュールおよび追加されたモジュールを検出する第１の修正モジュール検出部１２１と、修正されたモジュールを検出する第２の修正モジュール検出部１２２とが設けられている。

【００２８】本実施例のソフトウェアテスト支援システムの処理は、（１）テストケースを自動生成する処理と、（２）テストの十分性を計測・評価する処理と、（３）再テストを支援する処理とに分かれる。

【００２９】テストケースの自動生成処理は、ソースコード解析部１０とテストケース作成部２０と、テストケースレポート出力部３０が関係する。また、テストの十分性計測・評価処理は、テストプローブ挿入部４０、ロードモジュール作成部５０、テストカバレジ計測部６０、モジュール網羅率評価部７０、テストケース網羅率評価部８０、モジュール網羅率出力部９０およびテストケース網羅率出力部１００が関係する。さらに、再テスト支援処理は、修正ソースコード解析部１１０、ソースコード変更影響評価部１２０および再実行対象テストケース特定部１３０が関係する。以下、各処理について説明する。

【００３０】（１）テストケース自動生成処理

テストケースを自動生成するために、まずソースコード解析部１０を実行してソースコードの解析を行う。ソースコード解析部１０は、テストの対象となるソフトウェアのソースコード１を解析し、その内部構造を明らかにする役割を持つ。ここでソフトウェアの内部構造とは、ソフトウェアを構成するタスク、モジュールの関連を意味する。ソフトウェアがどのようなタスクから構成され、またどのようなモジュールによってタスクが構成され、それぞれのタスク、モジュールがどのような呼び合い関係にあるかを明確にするものである。例えば、ソースコード１がＣ言語によって作成されている場合は、制御構造（ｉｆ～ｔｈｅｎ～ｅｌｓｅ，ｗｈｉｌｅ，ｄｅ～ｗｈｉｌｅ，ｓｗｉｔｃｈ，ｇｏｔｏ，ｌａｂｅｌ，ｂｒｅａｋ，ｃｏｎｔｉｎｕｅ，ｒｅｔｕｒｎ，ｆｏｒ）などを解析して、構文規則に基づいてタスク、モジュールの解析を行うのである。関連する文献には、特開平１－２８３６３１公報などがある。

【００３１】図４は、ソースコード１の内部構造をソースコード解析部１０によって解析した例を示す図である。この例では、ソースコード１はモジュールＡ，Ｂ１，Ｂ２，Ｃ１，Ｃ２，Ｃ３，Ｄ１，Ｄ２から構成されており、例えば、モジュールＣ２はその下位モジュールとしてモジュールＤ１，Ｄ２を順次呼んでいることが判る。また、モジュールＣ３はモジュールＢ１から呼ばれた場合には単独で処理を行い下位のモジュールを呼ばないが、モジュールＢ２から呼ばれた場合にはモジュールＤ２をさらに呼んで処理を行うことが判る。従って、ソースコード解析部１０により得られるモジュール呼び合い関係は、モジュール内部の処理条件までを考慮した実際の呼び合い関係に相当する。

【００３２】ソースコード解析部１０の解析によってソフトウェアの内部構造を明らかにした後にテストケース作成部２０を実行して、ソフトウェアの内部構造をもとにソフトウェアテストとして実施すべきテストケースを作成する。テストケースは、ソフトウェアを構成するタスク、モジュールの呼び合い関係を考慮し、実際のソフトウェア操作で起こり得る呼び合い系列全てを網羅するように作成される。

【００３３】図５は、上記の考えに基づき、図４で示した内部構造を持つソフトウェアに関して、テストケースを抽出したものである。同図に示すように、Ｃａｓｅ－１からＣａｓｅ－９まで９通りのテストケースが抽出されている。次に、テストケースの抽出についての具体例として、例えば、画面処理を中心とするような事務処理系のソフトウェアについて考える。この場合、個々のモジュールはそれぞれの表示画面に対応する構造となる。図６は画面処理を中心とした財務事務処理関係のソフトウェアの内部構造を示すモジュール関連図の一例である。また、図７はこのソフトウェアの画面間の呼び合いに基づいたソフトウェアの処理操作の関連を示したブロック図である。例えば、このソフトウェアを実行した際に表示される図７の「会計管理システム初期画面」は、ソフトウェアの内部構造の観点からは図６に示す（会計処理メイン）モジュールの処理が対応する。図７において「会計管理システム初期画面」からは、「入力処理選択機能」「日次処理選択機能」等の下位機能の選択が可能であり、この「会計管理システム初期画面」からこれ

9

らの下位機能が呼ばれて実行されることが判る。これを図6のソフトウェア内部構造の観点からみると、（会計処理メイン）モジュールの下位にある（入力処理）、（日次処理）等の各モジュールを呼び出す形で実現されていることが判る。

【0034】この図6と図7を見ても明らかなように、このようなソフトウェアに関しては、内部構造を示すモジュール関連図は対象ソフトウェアの画面呼び合い関係や処理操作の関連に深く関係している。

【0035】従って、図6のモジュール関連図をもとに生成された上記のような呼び合い関係の遷移系列を網羅するテストケースは、対象ソフトウェアに関する処理操作の系列を網羅するものとなる。

【0036】図8（a）は、図6に基づいて生成されたテストケースの一例であり、一方、図8（b）は図7から想定される対象ソフトウェアの処理操作の流れを示したものである。図8（a）で生成されたテストケース（Ｃａｓｅ－2）では、モジュール呼び合い関係から、（総合管理メイン）モジュールから（会計管理メイン）モジューメを呼び、更に、（入力処理）（日次処理）（月次処理）（マスタメンテナンス）（決算処理）の各モジュールを順次呼ぶモジュール遷移系列がテストケースとして生成されている。これに対し、図8（b）で対象ソフトウェアの処理操作を考えると、（総合管理システム初期画面）から（会計管理システム初期画面）を選択し、更に下位の（入力処理機能選択画面）（日次処理選択画面）（月次処理選択画面）（マスタメンテナンス処理選択画面）（決算処理選択画面）を逐次起動操作する流れがあることが判る。この処理操作の流れは、図8（a）のＣａｓｅ－2で生成されたテストケースでのモジュール遷移に対応している。

【0037】このようにモジュール呼び合い関係の遷移系列から作成されるテストケースは、対象ソフトウェアに関する処理操作の系列を網羅している。なお、このモジュール関連図は、ソースコード解析部10の実行によって得られる。

【0038】このようにして作成されたテストケースは、テストケースファイル2として保持される。このテストケースファイル2は、テスト実施後にここで作成されたテストケースについてどのケースがテストされ、あるいはテストされなかったかの判定を行うための参考情報として利用される。また、1度目のテストで不具合が発生し、バグ修正を施したソフトウェアに対して再テストを行う場合にも、このテストケースファイル2に保持されたテストケースのうち有効なものが再利用される。

【0039】さらに、作成されたテストケースはテストケースレポート出力部30により、テストケース・レポート31として通常のプリンターから帳票形式で出力される。テストケース・レポート31は、ソフトウェアテストを実施する際の参考となる。図9は、帳票形式で出

10

力されたテストケース・レポート31の例を示す図である。

【0040】次に、モジュール関連図からのテストケース生成の手順を図10のフローチャートを用いて説明する。例えば、図4のモジュール関連図からテストケースを作成することを考えた場合、まず、テストケース作成の起点モジュールとして、モジュール関連図の最上位のモジュールであるモジュールＡが選択される（ステップ201）。次に、モジュール関連図により、この起点モジュールから呼ばれる下位モジュールが特定される（ステップ202）。図4の例では、モジュールＡの下位モジュールとして、モジュールＢ1，Ｂ2の2つのモジュールが特定される。

【0041】次に、このようにして特定された下位モジュールの呼出形式が確認される（ステップ203）。呼出形式とは、上位モジュールから下位モジュールが呼び出される方式をいう。下位モジュールの呼び出され方には、無条件に呼び出される場合と、条件に合ったときに呼び出される場合の2通りがある。例えば、図4の例では、起点モジュールであるモジュールＡの内部処理として、

・下位のモジュールＢ1，Ｂ2が順番に呼び出される場合と

・内部条件処理によって、下位のモジュールＢ1のみが呼び出される場合と

・同様に、下位のモジュールＢ2のみが呼び出される場合と

が考えられる。これらの下位モジュールの呼出形式の特定は、起点モジュールの内部処理手順を解析することによって行われる。

【0042】上記の操作によって、図11（a）に示すような起点モジュールを始点としたテストケース1次ツリーが作成される（ステップ204）。テストケース1次ツリーが作成された後に、現在対象としている起点モジュールより下位に呼出モジュールが存在するかどうかの判定を行う（ステップ205）。次に、起点モジュールを1つ下位のモジュールに移動する。即ち、図4の例では、起点モジュールとしてモジュールＢ1が選択される（ステップ206）。

【0043】その後、同様に選択されたテストケース作成の起点モジュールに対して、下位モジュールの特定と呼出形式が特定され、この起点モジュールに関するテストレース1次ツリーが作成される（ステップ204）。同様に、起点モジュールを順次下位に移動させることによって、各モジュールを起点とするテストケース1次ツリーが作成される（ステップ204）。図11（b）は図4に示したモジュール関連図をもとに作成された各モジュールに関するテストケース1次ツリーの例である。

このようにして作成された各モジュールのテストケース1次ツリーを融合して、網羅的にテストケースの作成が

11

行われる（ステップ207）。

【0044】図12（c）は、図11（b）の各モジュールをテストケース1次ツリーを融合して作成したテストケースの一例である。テストケース1次ツリーの融合では、ツリー上の同一ノードに着目して融合を進める。例えば、図12（c）に示したCase－1を融合生成するためには、以下の4つのテストケース1次ツリーに着目する。

【0045】・まずテストケース1次ツリー〈a〉の終端のモジュールB1に着目する。

・次にモジュールB1を始点とするテストケース1次ツリーを検索し、該当する1次ツリーとして〈b1〉を得る。

【0046】・テストケース1次ツリー〈b1〉の終端モジュールC1に着目する。

【0047】・次にモジュールC1を始点とするテストケース1次ツリーを検索し、該当する1次ツリー〈b2〉を得る。

【0048】・テストケース1次ツリー〈b2〉の終端モジュールD2に着目する。

【0049】・次にモジュールD2を始点とするテストケース1次ツリーを検索し、該当する1次ツリー〈b3〉を得る。

【0050】・テストケース1次ツリー〈b3〉の終端は、（E）として下位モジュールがないことが分かるため、ここでテストケース1次ツリーの探索・融合は完了する。

【0051】このようにして、テストケース1次ツリーの終端モジュールをキーに順次、探索・融合が進められる。上記の結果から以下に示すように、Case－1に示した一連のテストケースが融合獲得される。

【0052】

〈a〉　モジュールA→モジュールB1

〈b1〉　モジュールB1→モジュールC1

〈b2〉　モジュールC1→モジュールD2

〈b3〉　モジュールD2→E

Case－1

モジュールA→モジュールB1→モジュールC1→モジュールD2

（2）テストの十分性計測・評価処理

テスト実施時のテスト十分性を計測するために、対象ソフトウェアにカバレジ計測用プローブを挿入する。この手順を、図13のフローチャートを用いて説明する。

【0053】テストプローブ挿入については、まず、ソースコード解析部10によって分類された構成モジュールの全てに番号付けを行う（ステップ301）。図14は図5に示したソフトウェアを構成するモジュールに関して各モジュールの番号付けを行った一例であり、各モジュール名と付けられた番号の対応テーブルの生成が行われる。

12

【0054】対象となるソースコードを構成するタスク、モジュールの先頭部にそれぞれのタスク、モジュールの通過チェックを行うためのカウンタ（テストプローブ）が挿入される（ステップ302）。このカウンタ（テストプローブ）は、図15に示すように、モジュール通過管理ファイル4に当該タスク、モジュールの番号を書き込む命令（File　Write命令）を組み合わせたものである。

【0055】このカウンタ（テストプローブ）は、当該タスク、モジュールが実行された場合に、その当該タスク、モジュール番号の値をモジュール通過管理ファイル4に書き込む役割を持つ。例えば、図5に示したソフトウェアにおいて、モジュールC3はモジュール番号“06”が割り付けられており、この場合、ソースコードのモジュールC3の先頭部分に図15に示すようなテストプローブが自動的に挿入される。

【0056】このカウンタ（テストプローブ）は、当該タスク、モジュールが実行された場合に、その当該タスク、モジュール番号の値をモジュール通過管理ファイル4に書き込む役割を持つ。例えば、図5に示したソフトウェアにおいて、モジュールC3はモジュール番号“06”が割り付けられており、この場合、ソースコードのモジュールC3の先頭部分に図15に示すようなテストプローブが自動的に挿入される。

【0057】このテストプローブが埋め込まれたソースコードをコンパイル・リンクすることにより、テストカバレジの計測、テストの十分性の評価が可能となる（ステップ303）。

【0058】テストカバレジは、上記のテストプローブを挿入したソフトウェアを実行することにより計測される。テストカバレジ計測部60は、テスト実施時にテスト対象となるソフトウェアのモジュール通過管理ファイル4をセットする。このテストカバレジ計測部60の処理手順を図16のフローチャートを用いて説明する。

【0059】テストカバレジ計測部60では、テスト起動されるソフトウェアの起動回数が常に監視されている（ステップ401）。そして、そのソフトウェアの起動が始めてかどうかの判定が行われる（ステップ402）。以前に起動されていないソフトウェアが始めて起動された場合には、テストカバレジ計測部60では、モジュール通過管理ファイル4が書き込み可能な状態で自動作成される（ステップ403）。また、すでに過去に起動されたことのあるソフトウェアである場合には、該当するモジュール通過管理ファイル4が書き込み可能な状態でオープンされる（ステップ404）。

【0060】このようにモジュール通過管理ファイル4がオープンされた状態で、対象ソフトウェアが実際に起動され、モジュール通過時のテストプローブからのモジュール通過情報の書き込みが行われる（ステップ405）。このモジュール通過管理ファイル4は、上記のテ

13

ストプローブを挿入したソフトウェアの最初の起動時にセットされ、そのソフトウェアが変更されない限りモジュールの通過履歴が記録され続ける。

【００６１】モジュール通過管理ファイル４のファイル内容の例を図１７に示す。このファイルには、テスト実施日時及び実施時のタスク／モジュール・パスが記録されている。図１７に示すように、モジュール通過管理ファイル４は、対象ロードモジュール指定部１８及びテスト通過モジュール記録部１９から構成される。対象ロードモジュール指定部１８には、テスト対象となるソフトウェアのロードモジュールの名称、作成日時等が記録される。また、テスト通過モジュール記録部１９には指定されたロードモジュールがテスト実行された日時、及び、その際にテストプローブに基づきカウントされた通過したタスク／モジュール番号が記録される。

【００６２】図１７の例では、テスト対象ロードモジュールは“ＥＳＱ－ＳＹＳ．ＥＸＥ”、作成日時は“９３－６－３　１５：１９”であり、試験実施日が“９３－６－５　１０：１４”のテストでは、モジュール番号“０１，０４，０９，０２，０１”という系列がテストされたことが記録されている。

【００６３】テストカバレジ計測部６０の実行によって、モジュール通過管理ファイル４にモジュールの通過履歴が記録されると、次に、モジュール網羅率評価部７０とテストケース網羅率評価部８０が実行され、テストの十分性の評価が行われる。

【００６４】まず、モジュール網羅率評価部７０では、図１４に示した構成モジュールリストに記録されたモジュールと、図１７に示したモジュール通過管理ファイル４に記録されたテスト実施モジュール番号との照合が行われ、対象ソフトウェアを構成するモジュールの内、何パーセントを通過したかが評価される。また、同時にテスト時に通過していないモジュールが特定され、リスト（未試験モジュールリスト）が作成される。図１４，図１７の事例では、モジュールＣ３，Ｃ５，Ｄ４は未通過であることが判る。

【００６５】次に、テストケース網羅率評価部８０では、図５に示したテストケースと図１７に示したモジュール通過管理ファイル４に記録されたテスト実施モジュール番号が照合され、考えられるテストパスの内、何パーセントがテストされたかが評価される。同時に、テスト未実施のテストパスが特定され、リスト（未試験モジュール遷移パスリスト）が作成される。図５，図１７の事例では、図５に示したテストケースのうち、Ｃａｓｅ－４，５，７，８，９のパスがテストされていないことが分かる。

【００６６】上記のような２つの観点からテストの十分性の評価が行われ、評価結果はテスト結果評価ファイル５に記憶されると共に、テストカバレジレポート９１および未試験モジュール遷移パスレポート１０１として、

14

本システムの利用者に提供される。

【００６７】テストカバレジレポート９１について、図１８，図１９にその事例を示す。

【００６８】図１８はテストカバレジレポート９１の一つである「モジュール網羅率評価レポート」の一例であり、対象ソフトウェアを構成するモジュール数およびテスト時に実行されたモジュール数からテストにおけるモジュール網羅率が算出・表記されている。また、図１９は「モジュール遷移パス網羅率評価レポート」の一例であり、対象ソフトウェアを構成するモジュール間で可能な遷移系列数に対して実際のテストされた遷移系列の数が示され、これによりモジュール遷移系列のテスト網羅度が算出・表記される。

【００６９】未試験モジュール遷移パスレポート１０１については、図２０，図２１にその事例を示す。図２０は「未試験モジュールレポート」の一例であり、テスト実行時に通過していないモジュール名称がリストアップされている。また、図２１は「未試験モジュール遷移パスレポート」の一例である。このレポートは、ソフトウェアを構成するモジュール間で可能なモジュール間の遷移系列のうち、実際のテストで確認されていないパスが、モジュール名の系列としてリストアップされている。

【００７０】なお、本支援システムにおけるテストの十分性評価は、対象ソフトウェアを複数回実行した場合の総計を用いて行われる。

【００７１】（３）再テスト支援処理

通常、ソフトウェアの開発においては、テスト段階で発見されたバグを修正して、再度テストを行う。また、バージョンアップの際には機能追加や機能変更によってソースコードの修正が行われ、再度のテストが必要となる。このような場合、修正されたソースコードのどの部分をテストする必要があるかを特定し、これに対応したテストケースを作成、テストすることが問題となる。

【００７２】本システムでは、これらの再テストに対するテストケース作成についても支援を行う。

【００７３】ソースコード変更影響評価部１２０は、通常の１回目のテストを実施したソースコードと、テスト結果を反映させて修正したソースコードとの比較を行う役割を持つ。ソースコード変更影響評価部１２０には、削除されたモジュールおよび追加されたモジュールを検出する第１の修正モジュール検出部１２１と、修正されたモジュールを検出する第２の修正モジュール検出部１２２とが設けられている。

【００７４】第１の修正モジュール検出部１２１では、変更後のソースコードに対してソースコード構成解析が行われ、図４に示すようなソフトウェア内部構造が明瞭にされ、先に解析されている変更前のソフトウェア内部構造との対比が行われる。これにより、どのモジュールが削除・追加され、また、どのモジュール間の呼び合い

15

関係が変更を受けたかが特定できる。

【0075】また、第2の修正モジュール検出部122では、変更前のソースコードと変更後のソースコードそれぞれについて、各構成モジュールのステップ数、条件文数、ループ数等の比較が行われ、これらが変わっている場合には、変更・修正を行ったモジュールとして特定できる。

【0076】図22は、第1の修正モジュール検出部121での処理事例を示したものである。変更前のソフトウェアではモジュールB1がモジュールC1，C2，C3を呼んでおり、また、モジュールC2はモジュールD1，D2を呼んでいた。これに対して、変更後のソフトウェアではモジュールB1，C1，C2のみを呼び、また、モジュールC2はモジュールD1，E1を呼んでおり、呼び合い関係が変更されていることが判る。

【0077】図23は、第2の修正モジュール検出部122での処理事例を示したものである。変更前と変更後のソフトウェアでは、モジュールB1，C2，D1，E1の内部構造が変更されていることが判る。

【0078】修正・変更を受けたソースコードの変更点を特定した後、テストケース変更影響評価部120によって変更前に生成されたテストケースの中で、これらのソースコード変更の影響を受けると考えられるテストケースを特定する。第1の修正モジュール検出部121で検出された追加モジュールと、第2の修正モジュール検出部122で検出された修正モジュールは、全て再テストの対象となる。

【0079】図22，図23に示したソースコード変更影響評価について考えると、変更前に生成されたテストケースのうち、次のものについてはソースコード変更の影響を受けていると判定される。

【0080】

Case－1：A－B1－C1－C2－C3

Case－2：A－B1－C2－D1－D2

逆に、次のCase－3はソースコードの変更の影響を受けていないと考えられるため、再テストの必要は少ないと判定される。

【0081】

Case－3：A－B2－C3－D2

再実行対象テストケース特定部130では、変更前のソースコードに対して作成されたテストケースについて、ソースコード変更の影響が及んでいるかどうかを特定する働きを持つ。再実行対象テストケース特定部130で特定されたテストケースの情報はテストケース作成部20に与えられる。そして、テストケース作成部20では、この情報に基づいて、再度、変更されたソースコード6のテストケースが作成される。

【0082】例えば、先に示した図22，図23の例では、上記のCase－1，2のテストケースが再テストの対象となり、一方、Case－3はソースコード変更

16

の影響を受けていないと判定されるため、再テストのテストケースからは除外される。

【0083】このようにして変更されたソースコードに対して再度テストを行った場合に、変更されたソフトウェアに対するテストの十分性（網羅率）や未実行パス、未実行モジュールが特定される。この場合、テスト十分性評価については、修正変更の影響を受けないモジュールやモジュール遷移パスも含めて、どの程度をテストしたかが判定されるものである。また、未実行モジュール、未実行モジュールパスについても、変更の影響を受けないモジュール、モジュール遷移パスも含めて指摘の対象となる。

【0084】

【発明の効果】以上説明したように、本発明のソフトウェアテスト支援システムによれば、効率的なソフトウェアテストを容易に行うことができる。つまり、通常の開発言語によって作成されたソフトウェアに関して、そのテストのためのテストケースを自動的に抽出することが可能となる。このレベルで生成されるテストケースは、対象ソフトウェアのソフトウェア・システムテストの際の参考として利用でき、これにより効率的に抜けのないテスト仕様書を作成できるようになる。

【0085】また、テストの十分性を計測するテストプローブを対象のソフトウェアのソースコードに挿入することにより、テスト実行時に対象ソフトウェアを構成するモジュールをどの程度テストしたかを評価する「モジュール網羅率」および各モジュール間の関連を考慮して実行可能なモジュールの遷移系列をどの程度テストしたかを評価する「モジュール遷移パス系列網羅率」の評価が可能となる。また、これらの評価結果をレポートとして確認することが可能となる。これにより、対象ソフトウェアのテスト実施時に、具体的にどのモジュールあるいはどのモジュール遷移パスがテストされていないかが明確になり、効率的なテスト実施及び評価が可能となる。このモジュール網羅率、モジュール遷移パス網羅率、未試験モジュールの特定、未試験モジュールパス特定によって、より確実なソフトウェア・システム試験が可能となる。

【0086】さらに、ソースコードからのテストケース生成及びテスト実施時の十分性評価により、ソフトウェアのテストの効率的実施が容易となる。また、これによってテストの抜けや片寄りが防止され、ソフトウェアの効果的なテストが可能となり、品質、生産性の向上が図れる。

【図面の簡単な説明】

【図1】本実施例に係るソフトウェア支援システムの構成を示すブロック図である。

【図2】本実施例に係るソフトウェア支援システムの構成を示すブロック図である。

【図3】本実施例に係るソフトウェア支援システムの構

17

成を示すブロック図である。

【図４】ソフトウェアテストを実施する対象となるソフトウェアのモジュール構成を示す図である。

【図５】抽出されたテストケースの例を示す図である。

【図６】画面処理を中心とした財務事務処理関係のソフトウェア内部構造を示すモジュール関連図である。

【図７】画面処理を中心とした財務事務処理関係のソフトウェアの処理操作の関連を示したブロック図である。

【図８】生成されたテストケースと処理操作の流れの一例を示す図である。

【図９】本装置により出力されるテストケース・レポートの例を示す図である。

【図１０】テストケース生成の手順を示すフローチャートである。

【図１１】テストケース生成のためのテストケースツリーを示す図である。

【図１２】テストケース生成のためのテストケースツリーを示す図である。

【図１３】テストプローブ挿入手順を示すフローチャートである。

【図１４】各モジュールの番号付けを行った一例を示す図である。

【図１５】テストプローブが挿入されたソースコードの例を示す図である。

【図１６】テストカバレジ計測部の処理手順を示すフローチャートである。

【図１７】モジュール通過管理ファイルのファイル内容の例を示す図である。

【図１８】テストカバレジレポートのうち、「モジュール網羅率」評価結果レポートの一例を示す図である。

【図１９】テストカバレジレポートのうち、「モジュール遷移パス系列網羅率」評価結果レポートの一例を示す図である。

18

【図２０】「未試験モジュールレポート」の一例を示す図である。

【図２１】「未試験モジュールパスレポート」の一例を示す図である。

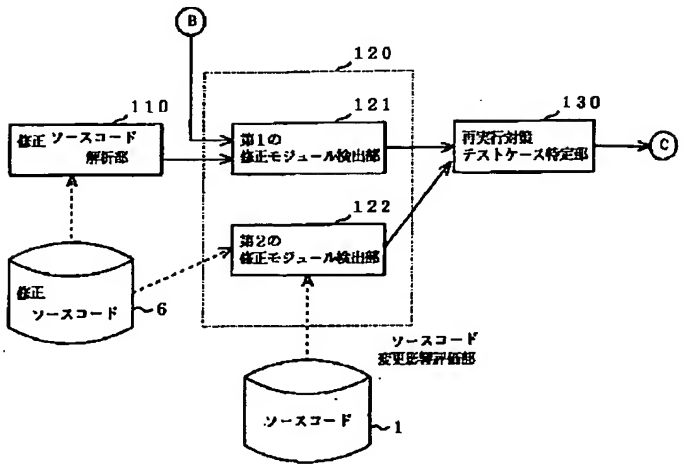
【図２２】再テスト時のソースコード変更影響評価部によるソフトウェア構成評価の一例を示す図である。

【図２３】再テスト時のソースコード変更影響評価による構成モジュール内部評価の一例を示す図である。

【符号の説明】

1…ソースコード
2…テストケースファイル
3…ロードモジュール
4…モジュール通過管理ファイル
5…テスト結果評価ファイル
6…修正ソースコード
１０…ソースコード解析部
２０…テストケース作成部
３０…テストケースレポート出力部
３１…テストケースレポート
４０…テストプローブ挿入部
５０…ロードモジュール作成部
６０…テストカバレジ計測部
７０…モジュール網羅率評価部
８０…テストケース網羅率評価部
９０…モジュール網羅率出力部
９１…テストカバレジレポート
１００…テストケース網羅率出力部
１０１…未試験モジュール遷移パスレポート
１１０…修正ソースコード解析部
１２０…ソースコード変更影響評価部
１２１…第１の修正モジュール検出部
１２２…第２の修正モジュール検出部
１３０…再実行対象テストケース特定部。

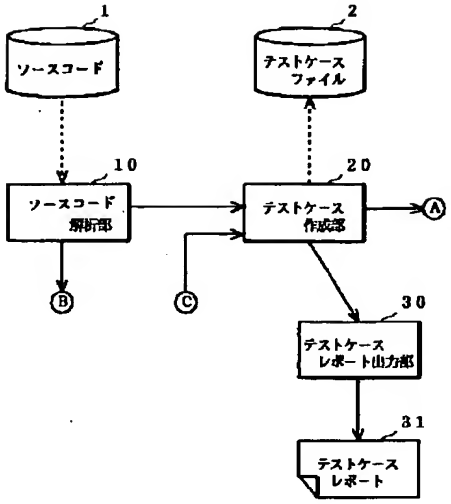
【図３】



【図５】

Case-1:	A－B１－C１－D１
Case-2:	A－B１－C２－D１
Case-3:	A－B１－C２－D２
Case-4:	A－B１－C３
Case-5:	A－B２－C３－D２
Case-6:	A－B２－C４－D３
Case-7:	A－B２－C５－D３
Case-8:	A－B２－C５－D４
Case-9:	A－B２－C５－D３－D４

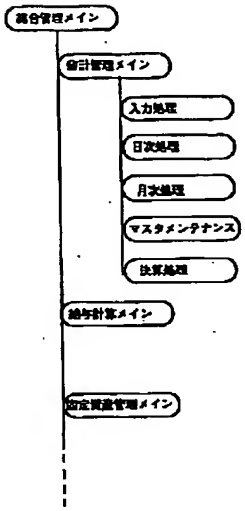
【図1】



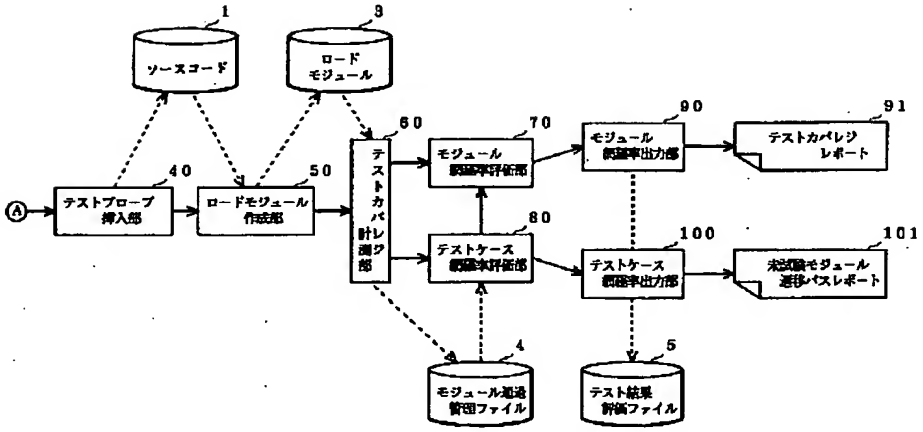
【図14】

モジュール名	モジュール番号
A	01
B1	02
B2	03
C1	04
C2	05
C3	06
C4	07
C5	08
D1	09
D2	0A
D3	0B
D4	0C

【図6】



【図2】

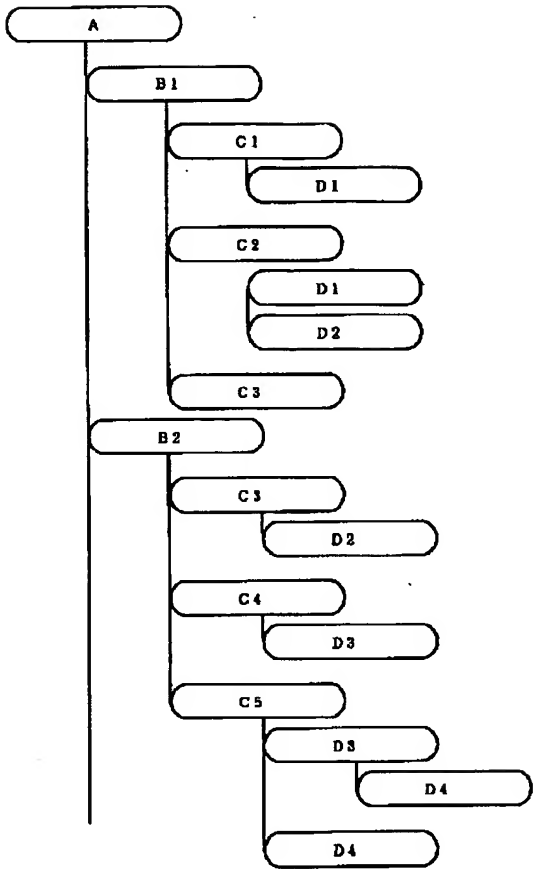


【図9】

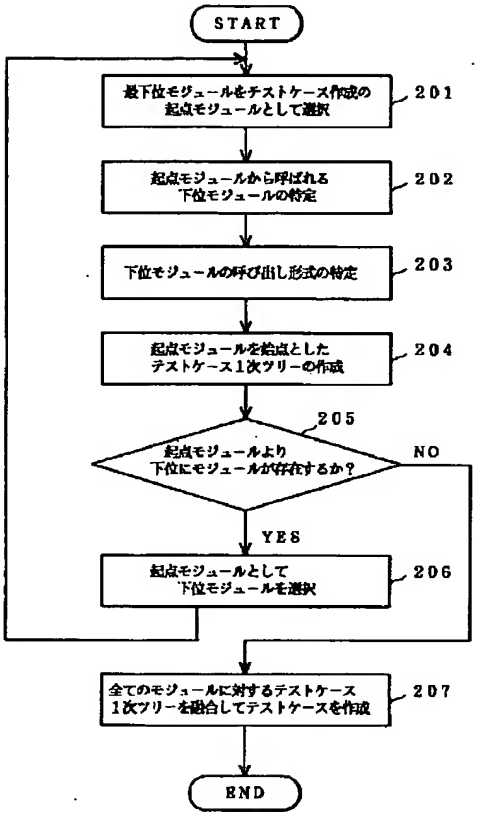
テストケース　レポート

Case-1:　A－B1－C1－D1
Case-2:　A－B1－C2－D1
Case-3:　A－B1－C2－D2
Case-4:　A－B1－C3
Case-5:　A－B2－C3－D2
Case-6:　A－B2－C4－D3
Case-7:　A－B2－C5－D3
Case-8:　A－B2－C5－D4
Case-9:　A－B2－C5－D3－D4

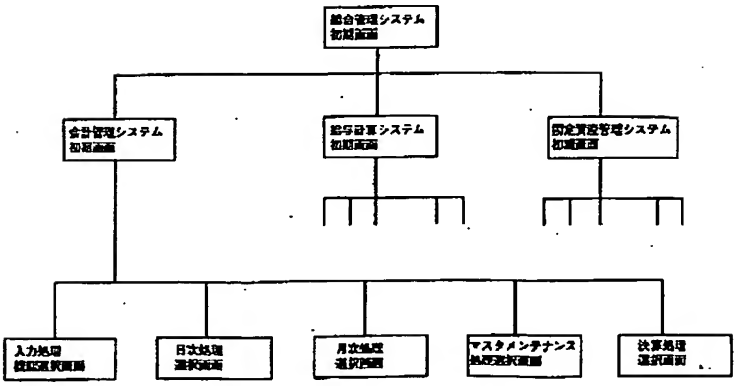
【図４】


A
B1
C1
D1
C2
D1
D2
C3
B2
C3
D2
C4
D3
C5
D3
D4
D4

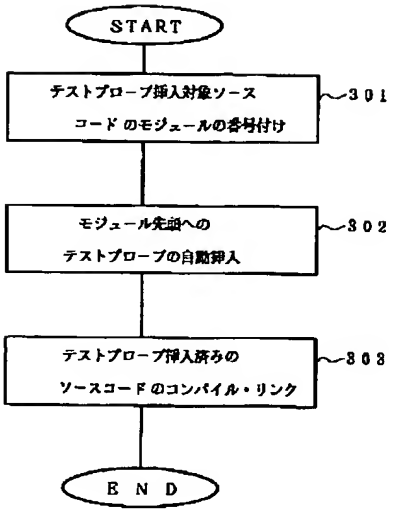
【図１０】


START
最下位モジュールをテストケース作成の起点モジュールとして選択
201
起点モジュールから呼ばれる下位モジュールの特定
202
下位モジュールの呼び出し形式の特定
203
起点モジュールを始点としたテストケース1次ツリーの作成
204
205
起点モジュールより下位にモジュールが存在するか？
NO
YES
起点モジュールとして下位モジュールを選択
206
全てのモジュールに対するテストケース1次ツリーを融合してテストケースを作成
207
END

【図７】


総合管理システム初期画面
会計管理システム初期画面
給与計算システム初期画面
固定資産管理システム初期画面
入力処理機能選択画面
日次処理選択画面
月次処理選択画面
マスタメンテナンス処理選択画面
決算処理選択画面

【図１３】


START
テストプローブ挿入対象ソースコードのモジュールの番号付け
301
モジュール先頭へのテストプローブの自動挿入
302
テストプローブ挿入済みのソースコードのコンパイル・リンク
303
E N D

【図８】

(a)

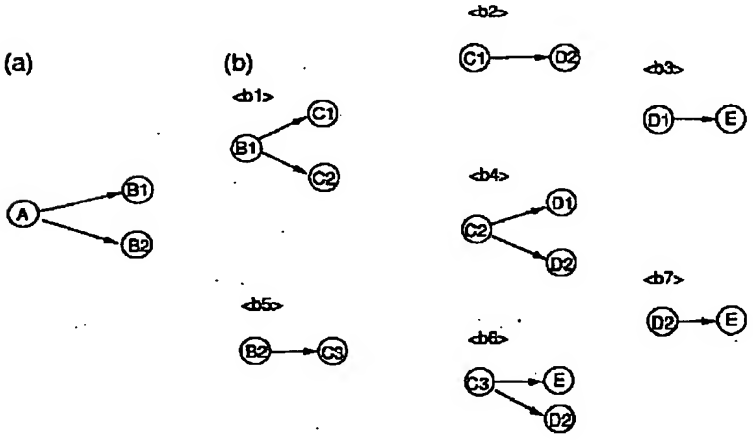
CASE-1: (総合管理メイン) － (会計処理メイン) －　(給与計算メイン) － (固定資産管理メイン)

CASE-2: (総合管理メイン) － (会計管理メイン) － (入力処理) － (日次処理) －
(月次処理) － (マスタメンテナンス) － (決算処理)

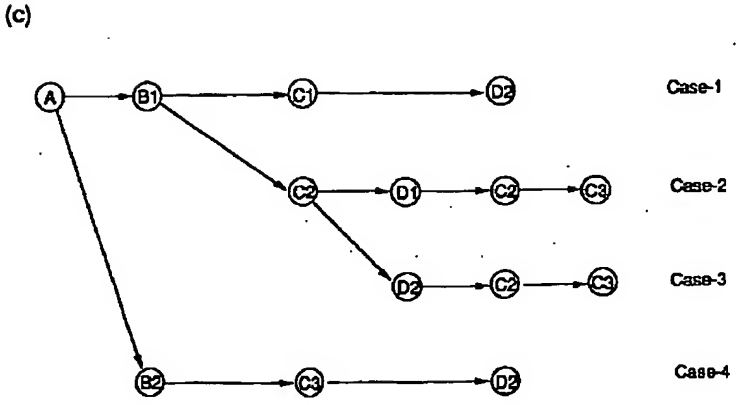
(b)

(総合管理システム初期画面) － (会計管理システム初期画面) － (入力処理機能選択画面)
－ (会計管理システム初期画面) － (日次処理選択画面)
－ (会計管理システム初期画面) － (月次処理選択画面)
－ (会計管理システム初期画面) － (マスタメンテナンス
処理選択画面)
－ (会計管理システム初期画面) － (決算処理選択画面)

【図１１】



【図１２】

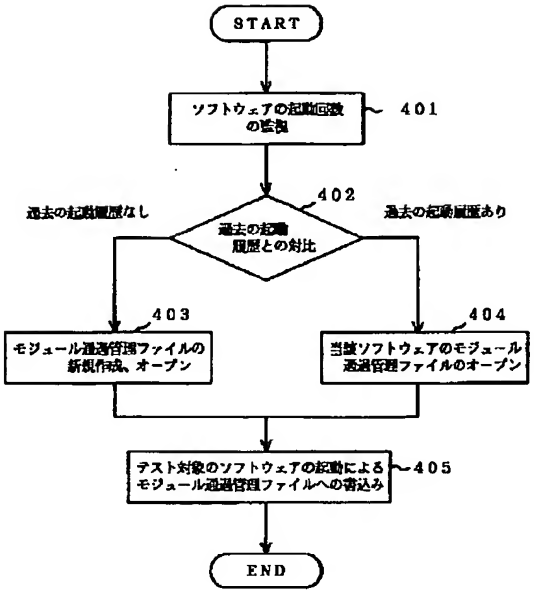


【図１５】

```
/*
   テストプローブ挿入事例　ソースコード
*/

# include  <process.h>
# include  'a_esqut.h'
# include  'aformat.h'

CS-main(argc,argv)
  int argc ;
  char *argv[] ;
{
  /*  テストプローブ挿入  */
    FILE *fopen(),*fp_w;
    int fd;
    int MOD_NO;
    fp_w =  fopen('TESTCOV',w);
    MOD_NO = 5;
    put(MOD_NO,fp_w);
    fclose(fp_w);
  /*  テストプローブ挿入終了    */

  static struct ANAINF anafnf ;
  int sts = 0;

  memset(&anainf,0x00,sizeof(anainf)) ;
  anainf.anabas.dmoflg =1 ;
  anainf.anabas/dmomsg =0 ;
  com_datesb(anainf.anabas.date) ;

  /*strcpy(anainf.anabas.dirbase,'YYKANA') ; */
  strcpy(anainf.anabas.dirbase,'YYESQUTYYSRCYYANA') ;
  system('cat -n YYesqutYYsample.trg') ;
  fprintf(stderr,'ana_tst : 計測開始の基準ディレクトリは[%s]です。
ＯＫ？>',anainf.anabas.dirbase) ;
  getchar() ;

/* com_exotim('ana_tst :  開始!'); */
  strcpy(anainf.trgfil.filnam,'SAMPLE.TRG') ;
  strcpy(anainf.trgfil.pthnam,'YYESQUTYY SAMPLE.TRG') ;
  strcpy(anainf.mtgfil.filnam,'SAMPLE.MTG') ;
  strcpy(anainf.mtgfil.pthnam,'YYESQDATYYSAMPLE.MTG') ;
  strcpy(anainf.rptfil.filnam,'SAMPLE.RPT') ;
  strcpy(anainf.rptfil.pthnam,'YYESQDATYYSAMPLE.RPT') ;
  strcpy(anainf.rpcfil.filnam,'SAMPLE.RPC') ;
  strcpy(anainf.rpcfil.pthnam,'YYESQDATYYSAMPLE.RPC') ;
  strcpy(anainf.errfil.filnam,'SAMPLE.ERR') ;
  strcpy(anainf.errfil.pthnam,'YYESQDATYYSAMPLE.ERR') ;
```

テストプローブ

【図１６】



【図１７】

18

対象ソフトウェア　　作成日
ESQ_SYS.EXE　　93-6-3　15:19

19

試験実施日	実施状況
93-6-5 10:14	01,02,04,09,04,02,01
93-6-5 10:55	01,02,04,02,05,0A,05,02,01
93-6-5 16:11	01,02,05,09,05,0A,05,02,01
93-6-9 09:34	01,03,07,08,07,02,01

【図１８】

モジュール網羅率評価

ロードモジュール名	モジュール数	試験通過モジュール	モジュール網羅率
ESQ-SYS.EXE	100	78	78
ESQ-ANA.EXE	150	75	50
DKB-LM.EXE	500	200	40

【図１９】

モジュール遷移パス　網羅率評価

ロードモジュール名	遷移系列数	試験実施系列数	網羅率
ESQ-SYS.EXE	63	42	66.7
ESQ-ANA.EXE	123	61	51.0

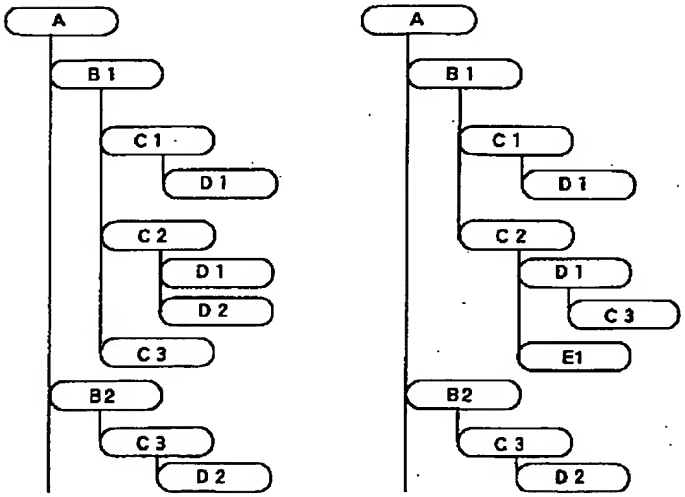
【図２０】

未通過モジュール一覧			
ロードモジュール名	未通過モジュール		
ESQ-SYS.EXE	trg_def()	pass_set()	conf_dir()
	conf_mod()	change_n()	ana_call()
ESQ-ANA.EXE	mod_ana()	cond_ana()	loop_ana()
	check-mod()		

【図２１】

未通過モジュール遷移パス一覧	
ロードモジュール名	未通過パス
ESQ-SYS.EXE	Case-4 : [main]-[sub_b1]-[mod_c3]-[mod_D2]
	Case-5 : [main]-[sub_b2]-[mod_c3]-[mod_D2]

【図２２】



【図２３】

Sorce Code : ESQ-ANA.C

	93-7-01			93-7-15		
	Step	Cond	Loop	Step	Cond	Loop
A	73	8	4	73	8	4
B1	35	6	3	32	4	2
B2	44	2	1	44	2	1
C1	109	33	9	109	33	9
C2	77	9	5	79	8	6
C3	22	0	1	22	0	1
D1	153	18	10	161	20	9
D2	54	7	6	54	7	6
E1	0	0	0	62	11	8